DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

AD
Acceso a Datos

UD 2
Manejo de Ficheros
Parte 3 - XML

IES BALMIS
Dpto Informática
Curso 2019-2020
Versión 1 (03/2019)

UD2 – Manejo de ficheros

ÍNDICE

11. Ficheros de XML – SAX (Lectura) y DOM (Lectura-Escritura)

Como probablemente sabrás, el formato XML (“eXtensible Markup Language”, lenguaje de marcas extensible) es un formato de texto delimitado por etiquetas, especialmente habitual para intercambio de información entre sistemas.

Existen varias alternativas para manipular ficheros XML desde Java.

- La primera forma, que no desarrollaremos, sería hacerlo "de manera artesanal", abriendo un fichero de texto y analizando cada elemento que se va leyendo, para ver dónde empieza cada etiqueta, dónde termina y cuál es el valor del elemento correspondiente.
- Una alternativa más eficiente es usar **SAX (Simple API for XML)**, que permite leer un fichero de forma secuencial (de principio a fin, sin conservar en memoria información sobre los datos que se habían leído anteriormente).
- Una tercera alternativa, que puede resultar más útil en muchos casos pero que consume mucha más memoria, es almacenar todo el árbol del documento (lo que se suele llamar el **DOM (Document Object Model)**.
- Pero existen más posibilidades, que no veremos. Por ejemplo, existen herramientas que son capaces de generar una estructura de clases Java nativas con la ayuda de hojas de estilo XSL.

Para los ejemplos que siguen, vamos a partir de un fichero XML que tenga el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<asignaturas>
  <asignatura id="ad">
    <nombre>Acceso a Datos</nombre>
    <cicloFormativo>DAM</cicloFormativo>
    <curso>Segundo</curso>
    <profesor>Ana</profesor>
  </asignatura>
  <asignatura id="int">
    <nombre>Interfaces</nombre>
    <cicloFormativo>DAM</cicloFormativo>
    <curso>Segundo</curso>
    <profesor>Juan</profesor>
  </asignatura>
  <asignatura id="lm">
    <nombre>Lenguajes de marcas</nombre>
    <cicloFormativo>ASIR</cicloFormativo>
    <curso>Primero</curso>
    <profesor>Sergio</profesor>
  </asignatura>
</asignaturas>
```

11.1.- Lectura de ficheros XML con SAX

SAX sólo permite lectura secuencial del fichero, de principio a fin, no se encarga de almacenar toda la estructura de datos en memoria. Por eso, será razonable emplearlo para análisis sencillos, en los que se desee volcar un fichero XML a otro tipo de fichero de datos o a una estructura dinámica en memoria creada por nosotros mismos.

La idea básica de SAX es que ciertos eventos (por ejemplo, la apertura o el cierre de una etiqueta XML) dispararán eventos que nuestro programa puede interceptar.

Los pasos serán:

- Crear un objeto de tipo "**SAXParserFactory**".
- A partir de él, crear un "**SAXParser**".
- Crear un manejador de eventos, un "**DefaultHandler**" y crear el contenido para los métodos
  - "**startElement**" (apertura de etiqueta),
  - "**characters**" (recepción del texto contenido entre la apertura y el cierre de una etiqueta) y
  - "**endElement**" (cierre de etiqueta).
- Llamar al método "**parse**" del parser, indicándole el fichero que debe analizar y el manejador de eventos que ha de emplear.

Así, un ejemplo del uso de SAX para recorrer el fichero anterior podría ser:

ReadXml1.java - Manejador

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ReadXml1 extends DefaultHandler {
  String etiquetaActual = "";
  String contenido = "";

  // Método que se llama al encontrar inicio de etiqueta: '<'
  public void startElement(String uri, String localName,
        String qName, Attributes attributes) throws SAXException {
    // Si el nombre es "asignatura", empieza una nueva y mostramos su id
    // Si no, memorizamos el nombre para mostrar después
    etiquetaActual = qName;
    if (etiquetaActual.equals("asignatura"))
        System.out.println("Asignatura: " + attributes.getValue("id"));
  }

  // Obtiene los datos entre '<' y '>'
  public void characters(char ch[], int start, int length) throws SAXException {
    contenido = new String(ch, start, length);
  }

  // Llamado al encontrar un fin de etiqueta: '>'
  public void endElement(String uri, String localName, String qName)
        throws SAXException {
    if (etiquetaActual.isEmpty()) {
      System.out.println(" " + etiquetaActual + ": " + contenido);
      etiquetaActual = "";
    }
  }
};
```

Sax1.java - Manejador

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;


public class Sax1 {

  public static void main(String[] args)
              throws ParserConfigurationException, SAXException, IOException {

    SAXParserFactory saxParserfactory = SAXParserFactory.newInstance();
    SAXParser saxParser = saxParserfactory.newSAXParser();
    File file = new File("./datos/asignaturas.xml");

    ReadXml1 handler = new ReadXml1();
    saxParser.parse(file, handler);

  }

}
```

De la misma forma, podemos "**parsear**" (o realizar un "parsing") un archivo XML para en vez de mostrarlo por pantalla, guardarlo en objetos, es decir "**mapear**" (o realizar un "mapping").

Las librerías que utilizamos son:

```
javax.xml.parsers
org.xml.sax
```

En el siguiente ejemplo, deseamos cargar un archivo XML que contiene versiones de Android, en un ArrayList de objetos de Java de la clase Version. Los datos son los siguientes:

```
+--------+--------------------+------+
| numero | nombre             | api  |
+--------+--------------------+------+
| 3      | Cupcake            | 1.5  |
| 4      | Donut              | 1.6  |
| 5      | Eclair             | 2.0  |
| 8      | Froyo              | 2.2  |
| 9      | Gingerbread        | 2.3  |
| 11     | Honercom           | 3.0  |
| 14     | Ice Cream Sandwich | 4.0  |
| 16     | Jelly Bean         | 4.1  |
+--------+--------------------+------+
```

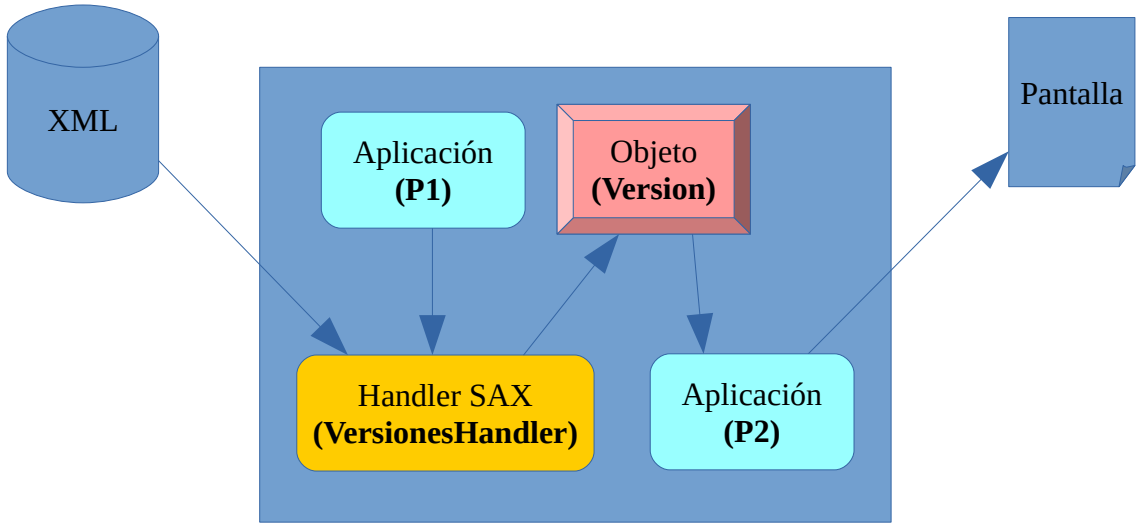
El esquema de la aplicación sería:

Aplicación → Handler SAX

Aplicación PARSER de un XML utilizando
un manejador (handler) SAX

El Parser mapea el XML en objetos Version
La Aplicación genera la Salida en Pantalla

VersionesSAX



Lo primero es crear el archivo XML de versiones teniendo en cuenta que el campo número, al ser la clave, deseamos que sea un atributo:

versiones.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<versiones>
	<version numero="1.5">
		<nombre>Cupcake</nombre>
		<api>3</api>
	</version>
	<version numero="1.6">
		<nombre>Donut</nombre>
		<api>4</api>
	</version>
	<version numero="2.0">
		<nombre>Eclair</nombre>
		<api>5</api>
	</version>
	<version numero="2.2">
		<nombre>Froyo</nombre>
		<api>8</api>
	</version>
	<version numero="2.3">
		<nombre>Gingerbread</nombre>
		<api>9</api>
	</version>
	<version numero="3.0">
		<nombre>Honercom</nombre>
		<api>11</api>
	</version>
	<version numero="4.0">
		<nombre>Ice Cream Sandwich</nombre>
		<api>14</api>
	</version>
	<version numero="4.1">
		<nombre>Jelly Bean</nombre>
		<api>16</api>
	</version>
</versiones>
```

Para poder almacenar esta información en un objeto Java deberemos crearnos una Clase:

Versiones.java

```java
public class Version {
  private double numero;
  private String nombre;
  private int api;

  public Version() {
  }

      ... getters y setters

  @Override
  public String toString() {
    return "Version{ numero=" + numero + ", nombre=" + nombre + ", api=" + api + '}';
  }

}
```

Ahora deberemos crear el **Manejador ("Handler")** que al parsear el fichero XML irá guardando en un objecto **ArrayList<Version>** los datos:

VersionesHandler.java

```
import java.util.ArrayList;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;


public class VersionesHandler extends DefaultHandler {
    private final ArrayList<Version> versiones = new ArrayList();
    private Version version;
    private String buffer;

    public ArrayList<Version> getVersiones() {
        return versiones;
    }

    @Override
    public void startElement(String uri, String localName, String qName,
                                Attributes attributes) throws SAXException {
        switch(qName) {
            case "versiones":
                break;
            case "version":
                version = new Version();
                versiones.add(version);
                version.setNumero(Double.parseDouble(attributes.getValue("numero")));
                break;
            case "nombre":
                buffer=""; // Vaciamos nuestro buffer
                break;
            case "api":
                buffer=""; // Vaciamos nuestro buffer
                break;
        }
    }

    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
        buffer = new String(ch,start,length); // Almacenamos en nuestro buffer
    }

    @Override
    public void endElement(String uri, String localName, String qName)
                                throws SAXException {
        switch(qName) {
            case "versiones":
                break;
            case "version":
                break;
            case "nombre":
                // Almacenamos en objeto
                version.setNombre(buffer);
                break;
            case "api":
                // Almacenamos en objeto
                version.setApi(Integer.parseInt(buffer));
                break;
        }
    }

}
```

Solo queda crear el código principal de la aplicación para que llame al "**Parser**":

VersionesSax.java

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;


public class VersionesSAX {

  public static void main(String[] args)
              throws ParserConfigurationException, SAXException, IOException {

        // P1 - Parsear el contenido del XML a objectos de la Clase Version
        SAXParserFactory saxParserfactory = SAXParserFactory.newInstance();
        SAXParser saxParser = saxParserfactory.newSAXParser();
        VersionesHandler handler = new VersionesHandler();

        File file = new File("./datos/versiones.xml");

        saxParser.parse(file, handler); // Llama al Manejador


        // P2 - Mostrar en pantalla los objetos de la Clase Version
        ArrayList<Version> versiones = handler.getVersiones();

        for (Version objVersion: versiones) {
          System.out.println(objVersion);
        }

        /*
        // Otra opción para recorrer un ArrayList
        versiones.forEach((v) -> {
            System.out.println(v);
        });
        */

    }

}
```

11.2.- Lectura y escritura de ficheros XML con DOM

Otra alternativa es DOM (Document Object Model).

DOM (Document Object Model)
https://es.wikipedia.org/wiki/Document_Object_Model

Si se prefiere memorizar todo el árbol del documento, de modo que se pueda acceder a cualquier dato en cualquier momento, la alternativa es emplear el DOM.

En este caso, los pasos serán:

- Crear un objeto de tipo "**DocumentBuilderFactory**".
- A partir de él, crear un "**DocumentBuilder**".
- Crear también un "**Document**" y, con "**parse**", indicarle el nombre del fichero que debe analizar.
- Obtener la lista de nodos con "**getElementsByTagName**", que se podrá recorrer con "for".
- Si ese nodo tiene atributos, se podrán obtener con "**getAttribute(nombre)**".
- Sus elementos estarán accesibles con "**getElementsByTagName(nombre)**"

La estructura al leer un XML con el modelo de objectos DOM sería:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Nodelist-NodeName>
	<Node Attribute="valueAttribute">
		<Element>valueElement</Element>
		<Element>valueElement</Element>
	</Node>
	<Node Attribute="valueAttribute">
		<Element>valueElement</Element>
		<Element>valueElement</Element>
	</Node>
	<Node Attribute="valueAttribute">
		<Element>valueElement</Element>
		<Element>valueElement</Element>
	</Node>
</Nodelist>
```

Un fuente completo de ejemplo podría ser así:

```
import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

public class Dom1 {
  public static void main(String[] args) {

    try {
      // P1 – Leer y cargar XML en objecto DOM
      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      dbFactory.setIgnoringComments(true);
      dbFactory.setIgnoringElementContentWhitespace(true);

      File inputFile = new File("asignaturas.xml");
      Document doc = dBuilder.parse(inputFile);
      doc.getDocumentElement().normalize();
          // normalize – elimina nodos vacíos, y une nodos de texto adyacentes
          //                 pero que estaban separados por intros

      // P2 – Recorrer DOM mostrando elementos
      System.out.println("Elemento base : " + doc.getDocumentElement().getNodeName());
      NodeList nList = doc.getElementsByTagName("asignatura");

      System.out.println();
      System.out.println("Recorriendo asignaturas...");

      for (int temp = 0; temp < nList.getLength(); temp++) {
        Node nNode = nList.item(temp);
        if (nNode.getNodeType() == Node.ELEMENT_NODE) {
          Element eElement = (Element) nNode;

          System.out.println("Codigo: " + eElement.getAttribute("id"));

          System.out.println("Nombre: " +
             eElement.getElementsByTagName("nombre").item(0).getTextContent());

          System.out.println("Ciclo: " +
             eElement.getElementsByTagName("cicloFormativo").item(0).getTextContent());

          System.out.println("Curso: " +
             eElement.getElementsByTagName("curso").item(0).getTextContent());

          System.out.println("Profesor: " +
             eElement.getElementsByTagName("profesor").item(0).getTextContent());

          System.out.println();
        } //if
      } //for

    } catch (Exception e) {
      e.printStackTrace();
    }

  }
}
```

Si deseamos escribir un XML teniendo en cuenta un objecto que utilice el modelo DOM, tendríamos que crear un **BuilderFactory**:

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = dbFactory.newDocumentBuilder();
dbFactory.setIgnoringComments(true);
dbFactory.setIgnoringElementContentWhitespace(true);
dbFactory = builder.newDocument();
```

El objecto **document** está vacío, y lo primero que debemos hacer es crear el objecto **raíz**. En nuestro caso crearemos productos:

```
Element productos = document.createElement("productos");
document.appendChild(productos);
```

Ahora crearíamos cada **elemento**. Crearemos un producto de muestra:

```
Element producto = document.createElement("producto");
productos.appendChild(producto);

producto.setAttribute("codigo", "1");

Element nombre = document.createElement("nombre");
nombre.appendChild(document.createTextNode("Teclado"));

producto.appendChild(nombre);
```

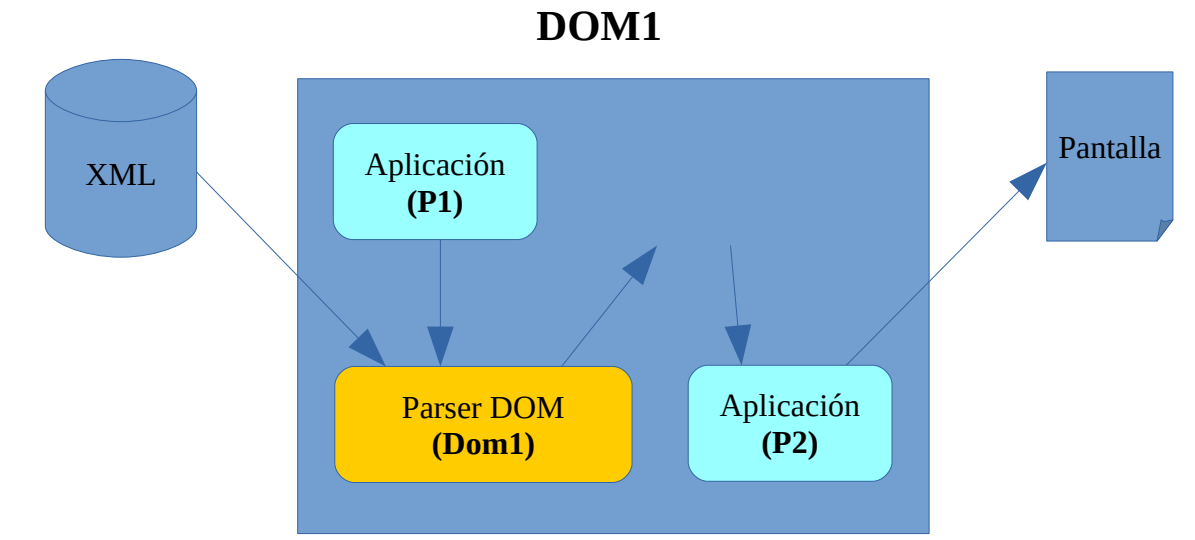
Por último, necesitaremos un **TransformerFactory** para crear el XML utilizando el objeto **document** y el fichero a crear.

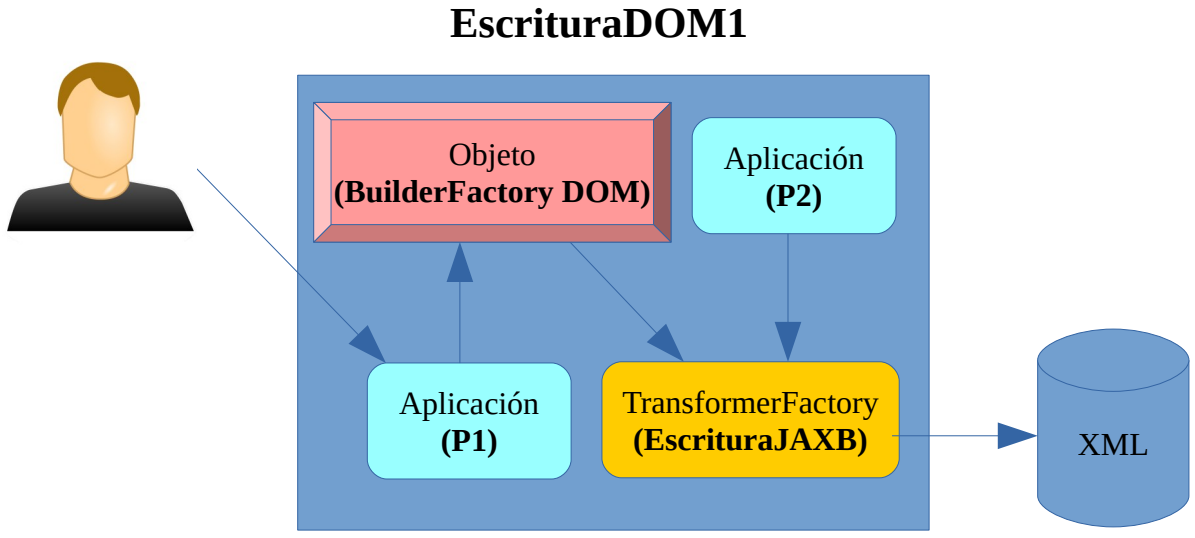
```
TransformerFactory factoria = TransformerFactory.newInstance();
Transformer transformer = factoria.newTransformer();

// Preparar fuente de "Object factory"
Source source = new DOMSource(document);

// Preparar salida para el archivo XML
File file = new File("./datos/productos.xml");
FileWriter fw = new FileWriter(file);
PrintWriter pw = new PrintWriter(fw);
Result result = new StreamResult(pw);

// Para tabulaciones
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

// Realizar transformación: "Object Factory" to "Archivo Serializado (XML")
transformer.transform(source, result);
```

Para que el código funcione necesitaremos los **import** correspondientes:

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

Fuente: Tech Krowd





12. Trabajo con ficheros XML

Probablemente hayas estudiado ya XML, bien porque hayas cursado el módulo **Lenguajes de marcas y sistemas de gestión de información**, o bien porque lo conozcas por tu cuenta. Si no conoces XML, te recomendamos que te familiarices con él, hay múltiples tutoriales y cursos en Internet.

El metalenguaje XML se crea para evitar problemas de interoperabilidad entre plataformas y redes. Con él se consigue un soporte estándar para el intercambio de datos: no sólo los datos están en un formato estándar sino también la forma de acceder a ellos. Y entre las ventajas de su uso destacamos:

- **Facilita el intercambio de información** entre distintas aplicaciones ya que se basa en estándares aceptados.
- **Proporciona una visión estructurada** de la información, lo que permite su posterior tratamiento de forma local.

12.1 Conceptos previos

¿Cómo se trabaja con datos XML desde el punto de vista del desarrollador de aplicaciones?

Una aplicación que consume información XML debe:

- Leer un fichero de texto codificado según dicho estándar.
- Cargar la información en memoria y, desde allí...
- Procesar esos datos para obtener unos resultados (que posiblemente también almacenará de forma persistente en otro fichero XML).

El proceso anterior se enmarca dentro de lo que se conoce en informática como "**parsing**" o **análisis léxico-sintáctico**, y los programas que lo llevan a cabo se denominan "parsers" o analizadores (léxico-sintácticos). Más específicamente podemos decir que el "parsing XML" es el proceso mediante el cual se lee y se analiza un documento XML para comprobar que está bien formado para, posteriormente, pasar el contenido de ese documento a una aplicación cliente que necesite consumir dicha información.

Schema: Un esquema (o schema) es una especificación XML que dicta los componentes permitidos de un documento XML y las relaciones entre los componentes. Por ejemplo, un esquema identifica los elementos que pueden aparecer en un documento XML, en qué orden deben aparecer, qué atributos pueden tener, y qué elementos son subordinados (esto es, son elementos hijos) para otros elementos. Un documento XML no tiene por qué tener un esquema, pero si lo tiene, debe atenerse a ese esquema para ser un documento XML válido.

12.2 Definiciones

Hay muchos "**parsers**" conocidos, como **SAX** (Simple API for XML). **SAX** es un API para parsear ficheros XML. Proporciona un mecanismo para leer datos de un documento XML.

¿Qué es y para qué sirve **JAXB** (Java Architecture for XML Binding)? JAXB simplifica el acceso a documentos XML representando la información obtenida de los documentos XML en un programa en formato Java, o sea, proporciona a los desarrolladores de aplicaciones Java, una forma rápida para vincular esquemas XML a representaciones Java.

JAXB proporciona métodos para, a partir de documentos XML, obtener árboles de contenido (generados en código Java), para después operar con ellos o manipular los los mismos en una aplicación Java y generar documentos XML con la estructura de los iniciales, pero ya modificados.

Parsear un documento XML consiste en "escanear" el documento y dividirlo o separarlo lógicamente en piezas discretas. El contenido parseado está entonces disponible para la aplicación.

Binding: Binding o vincular un esquema (schema) significa generar un conjunto de clases Java que representan el esquema.

Compilador de esquema o schema compiler: liga un esquema fuente a un conjunto de elementos de programa derivados. La vinculación se describe mediante un lenguaje de vinculación basado en XML.

Binding runtime framework: proporciona operaciones de unmarshalling y marshalling para acceder, manipular y validar contenido XML usando un esquema derivado o elementos de programa.

Marshalling: es un proceso de codificación de un objeto en un medio de almacenamiento, normalmente un fichero. Proporciona a una aplicación cliente la capacidad para convertir un árbol de objetos Java JAXB a ficheros XML. Por defecto, el marshaller usa codificación UTF-8 cuando genera los datos XML.

Unmarshalling: proporciona a una aplicación cliente la capacidad de convertir datos XML a objetos Java JAXB derivados.

12.3 Introducción a JAXB

JAXB permite mapear clases Java a representaciones en XML y viceversa mediante **anotaciones**.

JAXB proporciona dos principales características:

- La capacidad de serializar (marshalling) objetos Java a XML.
- Lo inverso, es decir, deserializar (unmarshalling) XML a objetos Java.

O sea que **JAXB permite almacenar y recuperar datos en memoria en cualquier formato XML, sin la necesidad de implementar un conjunto específico de rutinas XML** de carga y salvaguarda para la estructura de clases del programa.

El compilador de JAXB (schema compiler) permite generar una serie de clases Java que podrán ser llamadas desde nuestras aplicaciones a través de métodos sets y gets para obtener o establecer los datos de un documento XML.

El **funcionamiento esquemático** al usar JAXB sería:

- Crear un esquema (fichero .xsd) que contendrá las estructura de las clases que deseamos utilizar.
- Compilar con el JAXB compiler (bien con un IDE como NetBeans o desde línea de comandos con el comando xjc) ese fichero .xsd, de modo que nos producirá los POJOs, o sea, una clase por cada uno de los tipos que hayamos especificado en el fichero .xsd. Esto nos producirá los ficheros .java.
- Compilar esas clases java.
- Crear un documento XML: validado por su correspondiente esquema XML, o sea el fichero .xsd, se crea un árbol de objetos.
- Ahora se puede parsear el documento XML, accediendo a los métodos gets y sets del árbol de objetos generados por el proceso anterior. Así se podrá modificar o añadir datos.
- Después de realizar los cambios que se estimen, se realiza un proceso para sobrescribir el documento XML o crear un nuevo documento XML.

12.4 Funcionamiento de JAXB

Para construir una aplicación JAXB necesitamos tener un esquema XML.

Tras obtener el esquema XML, seguimos los siguientes pasos para construir la aplicación JAXB:

1) **Escribir el esquema**: es un documento XML que contiene la estructura que se tomará como indicaciones para construir las clases. Estas indicaciones pueden ser, por ejemplo, el tipo primitivo al que se debe unir un valor de atributo en la clase generada.
2) **Generar los ficheros fuente de Java**: para esto usamos el compilador de esquema, ya que éste toma el esquema como entrada de información. Cuando se haya compilado el código fuente, podremos escribir una aplicación basada en las clases que resulten.
3) **Construir el árbol de objetos Java**: con nuestra aplicación, se genera el árbol de objetos java, también llamado árbol de contenido, que representa los datos XML que son validados con el esquema. Hay dos formas de hacer esto:
   a) Instanciando las clases generadas.
   b) Invocando al método unmarshall de una clase generada y pasarlo en el documento. El método unmarshall toma un documento XML válido y construye una representación de árbol de objetos.
4) **Acceder al árbol de contenido** usando nuestra aplicación: ahora podemos acceder al árbol de contenido y modificar sus datos.
5) **Generar un documento XML** desde el árbol de contenido. Para poder hacerlo tenemos que invocar al método marshall sobre el objeto raíz del árbol.

Aunque estos pasos que acabamos de comentarte te parezcan algo complicados, vamos a ver un ejemplo sencillo, en el que clarificaremos todo esto, comprobando que no es tan difícil como parece.

Hay dos formas de generar un código usando JAXB

- Aprovechar el wizard (asistente), pero se necesita el esquema XML en formato XSD.
- Generar los archivos manualmente, donde solo se necesita el XML.

Para comprender el funcionamiento, es mucho mejor realizarlo manualmente.

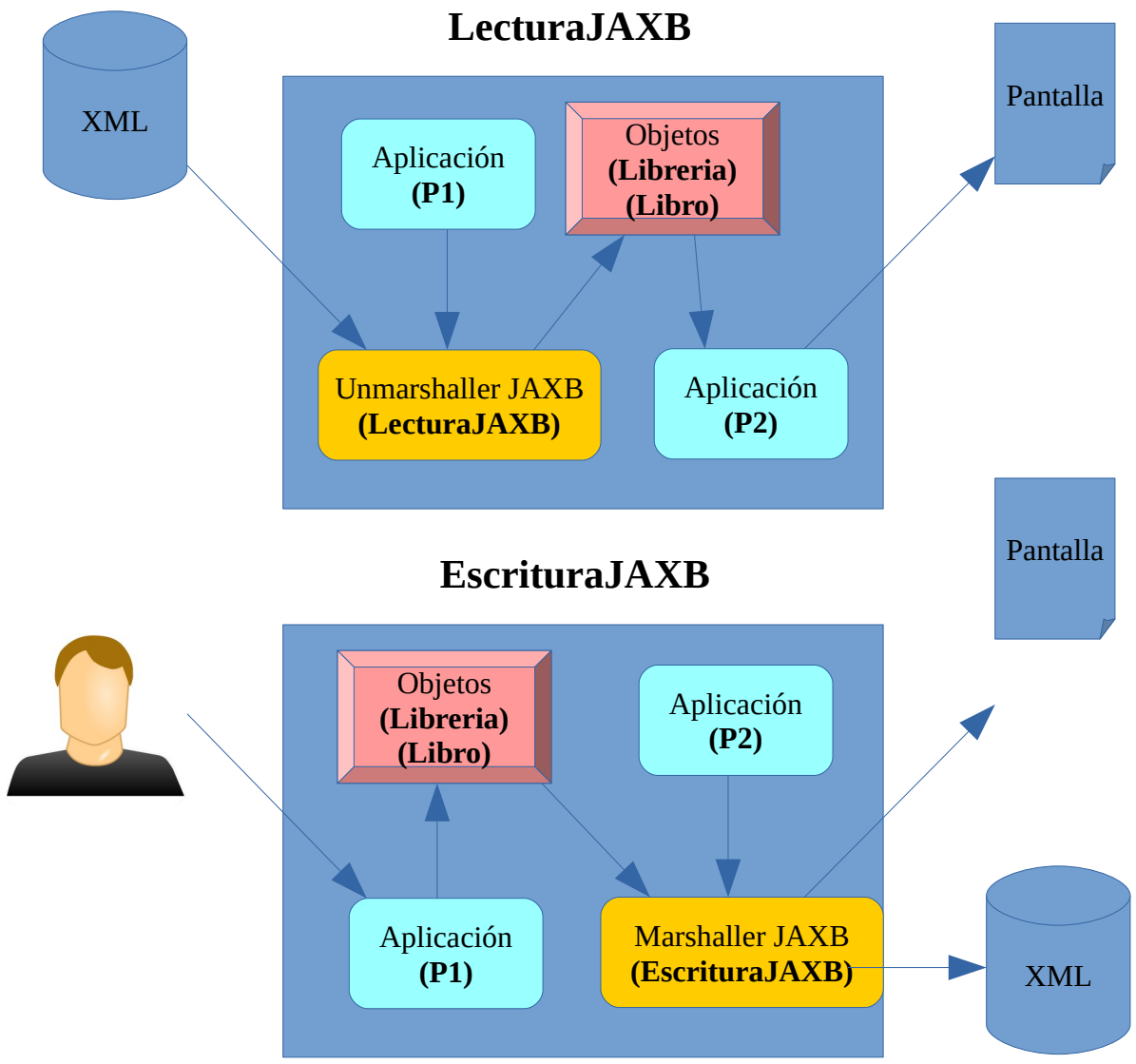
Los archivos a generar son:

- Clase (con anotaciones) que corresponderá con el conjunto de datos del XML
- Clase (con anotaciones) que corresponderá con los elementos existentes en el XML
- Java que llamará a ".unmarshal" para obtener el conjunto de datos.

En el ejemplo que vamos a mostrar, partiremos de una estructura de una librería que contiene libros en XML:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<libreria>
	<nombre>Libreria técnica</nombre>
	<libros>
		<libro isbn="9788441536258">
			<titulo>Java 8</titulo>
			<autor>Herbert Schildt</autor>
		</libro>
		<libro isbn="9788415033370">
			<titulo>Curso de programación web</titulo>
			<autor>Scott McCracken</autor>
		</libro>
		<libro isbn="9788426722126">
			<titulo>Python fácil</titulo>
			<autor>Arnaldo Pérez</autor>
		</libro>
	</libros>
</libreria>
```

Deseamos realizar una aplicación que utilice un mapeo a objetos utilizando JAXB.



Deberemos indicar en cada Clase la información del mapeo con las anotaciones.

Libro.java

```
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAccessType;

@XmlRootElement(name="libro")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder={"isbn","titulo","autor"})
public class Libro {
    @XmlAttribute(name="isbn")
    private String isbn;

    @XmlElement(name="titulo")
    private String titulo;

    @XmlElement(name="autor")
    private String autor;

    // *********************************
    // Constructor
    public Libro() {
    }

    // getters y setters
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }
    public void setAutor(String autor) {
        this.autor = autor;
    }
}
```

Libreria.java

```
import java.util.ArrayList;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAccessType;

@XmlRootElement(name="libreria")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder={"nombre","libros"})
public class Libreria {
    @XmlElement(name="nombre")
    private String nombre;

    @XmlElementWrapper(name="libros")
    @XmlElement(name="libro")
    private ArrayList<Libro> libros = new ArrayList<>();

    // *********************************
    // Constructor
    public Libreria() {
    }

    // getters y setters
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public ArrayList<Libro> getLibros() {
        return libros;
    }
    public void setLibros(ArrayList<Libro> libros) {
        this.libros = libros;
    }
}
```

La aplicación para leer utilizará **Unmarshaller** para mapear el XML en el objeto librería:

LecturaJAXB.java

```
import java.io.File;
import java.util.ArrayList;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

public class LecturaJAXB {
  public static void main(String[] args) throws JAXBException {

    // P1 - Unmarshaller de XML a libreria de libros
    JAXBContext context = JAXBContext.newInstance(Libreria.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    Libreria libreria =
             (Libreria) unmarshaller.unmarshal(new File("./datos/libreria.xml"));

    // P2 - Mostrar libros de la librería
    System.out.println("Nombre: "+libreria.getNombre());
    ArrayList<Libro> libros = libreria.getLibros();
    for (Libro lib : libros) {
      System.out.println(lib.getIsbn()+" - "+lib.getTitulo()+" - "+lib.getAutor());
    }
}
```

La aplicación para escribir primero deberá cargar unos libros de forma manual y luego utilizará **Marshaller** par escribir en el fichero

EscrituraJAXB.java

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

public class EscrituraJAXB {

  public static void main(String[] args) throws JAXBException, IOException {

    // P1 - Carga de objectos libros en libreria
    Libreria libreria = new Libreria();
    libreria.setNombre("Mi librería");

    ArrayList<Libro> libros = new ArrayList<>();

    Libro libro = new Libro();
    libro.setIsbn("123456789");
    libro.setTitulo("Libro 1");
    libro.setAutor("Autor 1");

    libros.add(libro);

    libro = new Libro();
    libro.setIsbn("023456789");
    libro.setTitulo("Libro 2");
    libro.setAutor("Autor 3");

    libros.add(libro);

    libreria.setLibros(libros);

    // P2 - Marshaller libreria de libros a --> arhivo de XML
    JAXBContext context = JAXBContext.newInstance(Libreria.class);
    Marshaller marshaller = context.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    marshaller.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");

    marshaller.marshal(libreria, System.out);
    // marshaller.marshal(libreria, new FileWriter("./datos/miLibreria.xml"));
    marshaller.marshal(libreria,
            new BufferedWriter(
                new OutputStreamWriter(
                   new FileOutputStream("./datos/miLibreria.xml", false),
                   StandardCharsets.UTF_8)));

    System.out.println("Archivo XML generado");

    System.out.println(libreria.getNombre());
  }
}
```

Fuente: Tech Krowd

Como podemos observar, en las clases Libro y Libreria se han utilizado anotaciones para indicar al Marshaller de JAXB como realizar el mapeo del XML a objetos de dichas clases.

Para comprender mejor el uso de estas anotaciones veamos la siguiente tabla:

EN CABECERA	
`@XmlRootElement(name="etiqueta")`	Establece la etiqueta raíz del archivo XML
`@XmlAccessorType(XmlAccessType.FIELD)`	Indica el lugar donde se escribe la anotación. FIELD define que se escriben al definir el campo. Aunque puede tomar otros valores distintos a FIELD, nosotros usaremos este.
`@XmlType(propOrder={"cpo1","cpo2"})`	Indica el orden de las etiquetas en el XML

DENTRO DE LA CLASE – ANTES DE CADA VARIABLES	

`@XmlElement(name="etiqueta")`
 	
`@XmlElement`
	Establece la etiqueta del elemento
	
	Si no se indica name se usará el nombre de la variable.
	
	Esta es la anotación por defecto si no se pone nada

`@XmlTransient`	Indica que esta vaiable no se escribe en el XML

`@XmlAttribute(name="etiqueta")`
 	
`@XmlAttribute`
	Define que la variable se escribirá como atributo en vez de como elemento
	
	Si no se indica name se usará el nombre de la variable.

`@XmlElementWrapper(name="etiqueta")`	Establece un nuevo nivel incluyendo la variable dentro de otra etiqueta nueva

DENTRO DE LA CLASE – EN CLASES ESPECIALES	
`@XmlAttribute`	Define que la variable se escribirá como atributo en vez de como elemento
`@XmlValue`	Establece que el valor de la variable se utilizará para el contenido de la etqueta de la clase
~~`@XmlElement`~~	No puede existir en este tipo de Clases

Veamos algunos ejemplos (lo que no aparece en negrita es opcional):

`<producto>`
`  <id>MA-1234</id>`
`</producto>`
	`@XmlRootElement(name="producto")`
	`@XmlAccessorType(XmlAccessType.FIELD)`
	`class Producto {`
	`  @XmlElement String id;`
	`}`

`<producto>`
`  <id>MA-1234</id>`
`  <nombre>Lámpara</nombre>`
`</producto>`
	`@XmlRootElement(name="producto")`
	`@XmlAccessorType(XmlAccessType.FIELD)`
	`@XmlType(propOrder={"id", "name"})`
	`class Producto {`
	`  @XmlElement(name="nombre") String name;`
	`  @XmlElement String id;`
	`}`

`<producto cod="MA-1234">`
`  <nombre>Lámpara</nombre>`
`</producto>`
	`@XmlRootElement(name="producto")`
	`@XmlAccessorType(XmlAccessType.FIELD)`
	`class Producto {`
	`  @XmlElement(name="nombre") String name;`
	`  @XmlAttribute(name="cod") String id;`
	`  @XmlTransient String modelo;`
	`}`

`<producto>`
`  <cod>MA-1234</cod>`
`  <nombre>Lámpara</nombre>`
`  <especif>`
`  60cm x 15cm roja`
`  </especif>`
`</producto>`
	`@XmlRootElement(name="producto")`
	`@XmlAccessorType(XmlAccessType.FIELD)`
	`@XmlType(propOrder={"id", "name", "especif"})`
	`class Producto {`
	`  @XmlElement(name="nombre") String name;`
	`  @XmlElement(name="cod") String id;`
	`  @XmlTransient String modelo;`
	`  @XmlList ArrayList<String> especif;`
	`}`
	`// especif.add("60cm x 15cm");`
	`// especif.add("roja");`

`<producto>`
`  <cod>MA-1234</cod>`
`  <nombre>Lámpara</nombre>`
`  <caracteristicas>`
`  <especif>`
`  60cm x 15cm`
`  </especif>`
`  <especif>`
`  roja`
`  </especif>`
`  </caracteristicas>`
`</producto>`
	`@XmlRootElement(name="producto")`
	`@XmlAccessorType(XmlAccessType.FIELD)`
	`@XmlType(propOrder={"id", "name", "especif"})`
	`class Producto {`
	`  @XmlElement(name="nombre")`
	`  String name;`
	
	`  @XmlElement(name="cod")`
	`  String id;`
	
	`  @XmlTransient`
	`  String modelo;`
	
	`  @XmlElementWrapper(name="caracteristicas")`
	`  @XmlElement(name="especif")`
	`  ArrayList<String> especif;`
	`}`
	`// especif.add("60cm x 15cm");`
	`// especif.add("roja");`